

[illegible]

Inventor:
Matthew Hendel
&
Kent Forschmiedt

ATTORNEY'S DOCKET NO. MS1-468US



TECHNICAL FIELD

This invention relates to computers and software, and more particularly to methods and arrangements for efficiently generating debugging information following a software failure.

BACKGROUND OF THE INVENTION

Computer failures can generally be traced to either hardware or software problems. This Background section discusses those failures that can be identified through a careful examination of the data stored by the computer at the time of failure.

When an operating system or application program crashes it is useful, if possible, to save information about the reasons for the crash and the state of the system when it crashed for later reference. Conventional techniques for collecting such POSTMORTEM INFORMATION (or DUMP INFORMATION) require an enormous amount of data be stored. For example, when an operating system (OS) crashes, the common technique of collecting postmortem information is to save the entire contents of the computer's RAM to permanent storage (e.g., disk, tape, floppy, etc.).

As computer's memory sizes and the amount of data associated with the OS continues to increase, the time it takes to store this postmortem information upon failure is correspondingly increased. Indeed, many users simply don't have time to generate postmortem information, and instead opt to manually reboot the computer, losing all postmortem debugging data. Consequently, problems may or may not be reported, and those that are reported would lack the critical data necessary to debug the problem. This means that a random, yet commonly

Consequently, the above stated problems and conventional solutions hamper the desire of many users and manufacturers for improved online support of the OS and applications. Here, for example, it would be unacceptable and potentially expensive for a user having a 28.8K Baud modem to transmit a 64 MB memory dump file to the manufacturer for postmortem analysis (it would take more than 5 hours).

As such, there is a need for improved methods and arrangements that substantially reduce the requisite amount of data required to conduct a significant postmortem analysis following an operating system or application failure. Preferably, the methods and arrangements will be advantageously configured to allow for online user support for a variety of users, computing devices, operating systems, applications, and the like.

SUMMARY OF THE INVENTION

Improved methods and arrangements are provided that substantially reduce the requisite amount of data required to conduct postmortem analysis following an operating system or application failure. The methods and arrangements can be advantageously configured to allow for rapid online user support for a variety of users, computing devices, operating systems, applications, and the like. The methods can be applied to either operating system failures or application program failures.

The various methods and arrangements address the generation of failure information when failures occur at either the operating system level (KERNEL-MODE) or in traditional user program (USER-MODE). For example, when a failure occurs to a program executing in USER-MODE a USER-MODE

1 MINIDUMP or USER MINIDUMP can be generated. Similarly, when a failure
2 occurs in kernel-mode a KERNEL-MODE MINIDUMP or KERNEL MINIDUMP
3 can be generated.

4 The above stated needs and others are met by a method that includes
5 determining when to generate a dump file, and generating a dump file by gathering
6 the thread, thread context and callstack for the thread that caused the failure; also
7 included in the dump are the process containing the failing thread and the reason
8 for the crash (e.g., invalid memory reference). The resulting dump file can then be
9 stored to a storage medium and accessed for subsequent analysis. In certain
10 implementations the dump file also includes information about multiple threads,
11 multiple processes or portions of data for the failing process. When configured as
12 a kernel minidump, the list of device drivers for the system is included.

13 To further support a kernel minidump, the method may also include
14 allocating a buffer space in memory during an initialization process, and reserving
15 space on a storage medium drive suitable for writing the contents of the buffer.
16 The method then includes generating the kernel minidump file by initially storing
17 the list of device drivers, crashing process and thread, as well as the thread-context
18 for the thread and the callstack associated with the thread. Additionally, the kernel
19 minidump contains information identifying the reason for generating the crash.
20 The method further includes copying the kernel memory dump file from the buffer
21 space to the storage medium as a kernel minidump. In still other implementations,
22 the method includes, upon re-initialization, accessing the kernel minidump on the
23 storage medium and using at least a portion of it to further understand the failure
24 that occurred.
25

1 The method is also applicable to non-operating system programs. Here, for
2 example, a user minidump file is generated by gathering the following information
3 at the time of a failure: the list of threads running at the time of the crash; the
4 thread-context and callstack for each running thread; the list of modules loaded at
5 the time of the crash; the reason for the crash; and, selected regions of the
6 process's memory that relate to the cause of the crash.

7 The above stated needs and others are met by an arrangement having a
8 system coupled to memory and at least one data storage device. The operating
9 system is configured to determine when to generate a dump file while running
10 computer instructions, and generate the dump file in the memory by gathering
11 callstack information for at least one running thread, thread context information
12 about the at least one running thread, process identifying information associated
13 with the at least one running thread, and information identifying the reason for
14 generating the dump file.

15 Methods for use in communicating between a client process and a server
16 process in a distributed processing system are also provided. Here, for example,
17 the methods can be used to provide application-programming interfaces (APIs) or
18 like capability when writing and reading dump files.

19 For a write dump operation, such methods include having the client process
20 issue a write dump file call having a plurality of call parameters including a
21 process handle, a process identifier, a handle to a file where dump file information
22 is to be written, and a dump type identifier. The server process receives this write
23 dump file call and parses the call to retrieve the parameters, and then in response,
24 issues a write dump file call acknowledgment providing a true-false indication.

For a read dump operation, such methods include having the client process issue a read dump file call having a plurality of call parameters including a header of a dump file and a data type identifier of data to read from a dump file, having the server process receiving the read dump file call and parsing the call to retrieve the parameters, and then having the server process issue a read dump file call acknowledgment providing a true-false indication and a plurality of call return parameters including a pointer to a beginning of a dump stream, and a stream size identifying the size of the dump stream. The plurality of call return parameters may also include a pointer to a dump file directory.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the various methods and arrangements of the present invention may be had by reference to the following detailed description when taken in conjunction with the accompanying drawings wherein:

Fig. 1 is a block diagram depicting an exemplary computer system suitable for use in generating a dump file.

Fig. 2 is a flow-chart depicting a kernel minidump process for use in generating postmortem debug information of an operating system failure as might be experienced from time to time in the computer system of Fig. 1.

Fig. 3 is a flow-chart depicting a user minidump process for use in generating user postmortem debug information of a computer application failure, for example, as might be experienced from time to time in the computer system of Fig. 1.

other optical media. The hard disk drive 27, magnetic disk drive 28 and optical disk drive 30 are each connected to bus 23 by applicable interfaces 32, 33 and 34, respectively.

The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for computer 20. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 29 and a removable optical disk 31, it should be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, random access memories (RAMs) read only memories (ROM), and the like, may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24, or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37, and program data 38. A user may enter commands and information into computer 20 through input devices such as keyboard 40 and pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are connected to the processing unit 21 through an interface 46 that is coupled to bus 23.

A monitor 47 or other type of display device is also connected to bus 23 via an interface, such as a video adapter 48. In addition to the monitor, personal computers typically include other peripheral output devices (not shown) such as speakers and printers.

Computer 20 can operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 50. Remote computer 50 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to computer 20. The logical connections depicted in Fig. 2 include a local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, computer 20 is connected to the local network 51 through a network interface or adapter 156. When used in a WAN networking environment, computer 20 typically includes a modem 54 or other means for establishing communications over the wide area network 52, such as the Internet. Modem 54, which may be internal or external, is connected to bus 23 via interface 46. In a networked environment, program modules depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Figure 6 depicts a simplified model of an Operating System. At the top of the diagram is the operating system's KERNEL. The kernel runs its own code and the code from any number of KERNEL DEVICE DRIVERS. Note that the arrow connecting the kernel to the device drivers is labeled "1 to N". This means that there can be many device drivers for every kernel. The kernel runs any number of PROCESSES. Each process contains a kernel portion and user portion. The kernel portion of a process is accessible only by the kernel and its device

1 drivers. The user portion is to both the user-mode process and kernel-mode
2 portion of the process.

3 The process's code is executed either by the kernel (KERNEL CODE) or
4 by the process itself (USER CODE). Processes can accomplish several tasks at
5 once. Each of these tasks is a THREAD. A thread, like a process, contains both a
6 user-mode portion, and a private kernel-mode portion. Each thread contains one
7 THREAD CONTEXT. A thread context specifies exactly what a thread is doing
8 at any specific moment. The thread context contains one CALLSTACK. Thus,
9 there is a kernel-mode only callstack and a user-mode callstack.

10 A flow-chart depicting an exemplary kernel-mode minidump process 100
11 for use in generating data suitable for conducting a postmortem analysis of a
12 computer failure is provided in Fig. 2.

13 As discussed below, kernel-mode minidump process 100 generates a very
14 compact representation of the running system information in the event of a system
15 failure. The resulting kernel minidump file is capable of being created and written
16 to disk very quickly, as well as quickly transported, for example, via e-mail over
17 the Internet or other communication resource. Despite its relatively small size, the
18 kernel minidump file contains enough information to allow a developer or product
19 support engineer to significantly reproduce the computing environment and
20 otherwise debug the faulting driver or operating system failure.

21 One advantage of kernel minidump process 100 is that it is capable of
22 directly addressing the reliability and stability of the operating system, since all
23 system failures can be logged and sent to the manufacturer for analysis. Thus, for
24 example, as a result of kernel minidump process 100, both device driver
25 developers and operating system developers can easily diagnose and quickly

respond to user issues that usually only occur in the field. Furthermore, since the resulting kernel minidump file is significantly compact enough it can also be added to a database of problems requiring further analysis or comparison.

In addition to traditional operating systems, it should be noted that kernel minidump process 100 is also a particularly desirable feature for enterprise-level operating systems, especially those operating systems that use a large number of third party drivers or have high reliability requirements.

Unlike previous solutions, kernel minidump process 100 is preferably designed to generate as minimal amount of information as necessary to meaningfully debug the failure. Thus, in accordance with certain exemplary implementations, kernel minidump process 100 generates a kernel minidump file essentially containing the processor information for the crashing processor, the thread for the crashing process, the process information for the crashing process, the callstack for the crashing thread, a list of device drivers loaded at the time of the crash including the location within the system where they are loaded and the version of the device that was loaded, and the specific cause for the crash.

In this example, kernel minidump process 100 does not store the actual code (i.e., executable instructions) for the running operating system or any of the loaded drivers. Instead, kernel minidump process 100 stores version information about the operating system and each driver that was loaded, and the virtual address where the driver was loaded. Then, during the subsequent analysis/debugging stage, the analyst can locate the proper driver version and load it into memory for debugging purposes at that time.

By not saving code, therefore, kernel minidump process 100 is able to generate a much more compact kernel minidump file. By way of example, in

1 certain exemplary implementations the kernel minidump file for Microsoft®
2 Windows® 2000 is only 64 KB, still contains sufficient information to locate the
3 crashing driver, generate a symbolic callstack for the driver and locate the source
4 code line and file where the driver failed.

5 In accordance with still further features, a user may also access certain local
6 variables and information about the cause of the crash. These may be helpful to
7 online or telephone support personnel or processes.

8 In step 102 of Fig. 2, at initialization/boot time, a kernel minidump buffer
9 capable of holding the required kernel minidump information is allocated along
10 with a kernel minidump file that is large enough to hold the kernel minidump
11 information. For example, in the Windows® 2000 example, the kernel minidump
12 file is allocated from the paging file. Also in step 102, a list of raw disk sectors
13 that form the kernel minidump file is obtained.

14 Next, in step 104 space for the disk drivers is reserved on the disk drive
15 where the kernel minidump file will be stored.

16 In step 106, when the computer system fails, the processor context of the
17 crashing processor is gathered from the system and written to the kernel minidump
18 buffer. In step 108, the thread information for the thread that was executing when
19 the failure occurred is gathered from the system and written to the kernel
20 minidump buffer. Similarly, in step 110, the process information for the process
21 that owns the crashing thread (above) is gathered from the system and written to
22 the kernel minidump buffer.

23 In step 112, the kernel stack for the crashing thread is gathered from the
24 system and written to the kernel minidump buffer.
25

In Step 114 the list of kernel-mode components, such as, the kernel, device drivers and hardware abstraction layers (HALs) that were loaded when the computer failure occurred and their respective load-addresses and appropriate version information (e.g., a checksum, timestamp, and image size) is gathered from the system and written to the kernel minidump buffer.

In step 116, specific information about the cause of the crash is gathered from the system and also written to the kernel minidump buffer. In Windows®, 2000, for example, this information is provided as the BugCheckCode along with the four BugCheck words associated with the failure.

In step 118, the kernel minidump information from steps 106 through 116 is transferred or otherwise copied from the kernel minidump buffer to disk. Since the file system is never invoked, faults in the file-system can be caught. In certain implementations, the resulting kernel minidump information is stored in an indexed table.

Next, in step 120, when the computer system is rebooted, the kernel minidump file is transferred from its temporary location, where it was written to by fault-handling code in step 118, to a more permanent location, for example, in a user-specified directory containing a list of kernel minidump files. These files can then be accessed, forwarded or otherwise handled as necessary to study the failure.

Those skilled in the art will recognize that the various steps in kernel minidump process 100 can be rearranged or reconfigured in many ways, as required to meet the needs of a given computing system environment.

One of the interesting aspects about the resulting kernel minidump file is what information can be left out. Thus, in the example above, no code for running components is saved (i.e., executable instructions), no dynamically allocated

memory is saved, and no initialized or uninitialized data is saved. Also, no information about threads, processes or processors except the crashing thread, process and processor is saved in the example above. While certain information may be necessary for other implementations, in the above example (Fig. 2) this appears to significantly minimize the amount of information required.

Kernel minidump process 100 is essentially directed towards system crashes. Thus, there remains a need for similar processes for non-system failures, i.e., user application failures.

Conventional postmortem debug information collection techniques tend to store all user-accessible memory for an application, including code for all user-mode shared libraries, dynamically allocated memory, and all stack data, upon the occurrence of a failure. This produces very large dump files, of which only a portion may be required to subsequently analyze the failure.

With this in mind, Fig. 3 depicts a user minidump process 200 for use in generating significantly smaller user-mode dump files containing selected information that is suitable for conducting postmortem analysis of an application failure.

In step 202, user minidump process 200 gathers information about the callstack for all running threads. In step 204, the exception information that caused the fault, if any, and thread context of the fault is gathered. A list of all loaded modules for the faulting application is gathered in step 206. In step 208, the processor context for all threads is gathered. In some configurations of minidump process 200, additional information about the crash is also saved at this point. In step 210 the results of steps 202 through 208 are written to disk.

1 In accordance with certain implementations, for example, user minidump
2 process 200 does not require instrumentation of the application, hooking of system
3 application programming interface (API) calls, or either local or remote symbolic
4 information to be available. Instead, the resulting user minidump file can be
5 generated by the client application simply using only information readily available
6 through standard debugging APIs, such as, for example, Win32 APIs.

7 The current implementation of the Dr. Watson tool in Windows® 2000, for
8 example, saves data for the entire process's address space. While this tends to
9 guarantee full postmortem analysis of the application, huge dump files are often
10 created (e.g., 50MB or more)

11 In contrast, using user minidump process 200, in the event that Microsoft®
12 Internet Explorer (IE) crashes, instead of a Dr. Watson dialog coming up, IE can
13 itself catch the error and write a user minidump file to disk. Since the user
14 minidump file size is relatively small (about 60 KB for IE), it takes only a second
15 or two to write the user minidump file to disk. When IE restarts, it could then note
16 that a user minidump file was written. Here, for example, IE may then access a
17 remote bug database that identifies that the problem experienced was a bug that
18 has since been fixed in a Service Pack release of IE. The user could then be
19 prompted to upgrade IE to fix the problem. In another illustrative scenario, assume
20 Microsoft® Excel crashes on a user's machine. As with the IE scenario, with user
21 minidump process 200 Excel can catch the exception and write a user minidump
22 file to disk. Here, the user may be notified that there was a program error and that
23 the use minidump file should be mailed to the developer. The minidump file can
24 then be e-mailed to developer where it can be forwarded to the development team
25 responsible for the code in Excel. Using standard debugging tools like Visual

Studio, the development team is able to walk the callstack, view parameters to functions, view local variables, and view the exact line and file where the failure occurred. Using this information the development team will likely be able to fix an obscure program error that otherwise would not have been reproducible.

In certain implementations of Windows® 2000, for example, two additional functions in the form of APIs are added to the DBGHELP system DLL. The first API is a write user minidump API (*WriteMiniDump*), and the second API is a read user minidump API (*ReadMiniDump*). These functions are configured to allow an arbitrary application to write and read a user minidump file. Experiments suggest that a user minidump file of approximately 100KB would be able to walk the callstack of all threads, read any active local variables, view parameters to functions, and view source code for the faulting application. Moreover, existing user-mode debugging tools, such as, Visual Studio and the like, can be easily debug a user minidump file. Furthermore, since the user minidump file format is completely specified and public, various developers can write custom applications to process the user minidump file as necessary.

Fig. 4 is a table 300 listing experimental results associated with user minidump process 200 as applied to certain applications during a failure. Here, user minidump files were created for Microsoft® Excel 2000, Outlook® 2000 and IE 5.0, using an x86 PC. As can be seen, the size of the resulting user minidump files for each of these applications is fairly small, ranging from 38 KB to 61 KB. Consequently, such user minidump files can be quickly saved and easily transported.

As shown in greater detail below, in certain implementations two system API calls have been added to DBGHELP of Windows® 2000, one to write a user

1 minidump file and one to read a user minidump file. Any application, including
2 postmortem tools, debuggers or the application itself will be able to write a user
3 minidump file at any time. In this manner, an application is explicitly allowed to
4 write a user minidump file even when there is not an exception or failure.

5 The *WriteMiniDump* API is:

6
7 BOOL
8 WINAPI
9 *WriteMiniDump*(
10 IN HANDLE *hProcess*,
11 IN DWORD *ProcessId*,
12 IN HANDLE *hFile*,
13 IN MINIDUMP_TYPE *DumpType*,
14 IN PMINIDUMP_EXCEPTION_INFORMATION *ExceptionParam*,
15 IN PMINIDUMP_USER_STREAM_INFORMATION *UserStream*,
16 IN PMINIDUMP_CALLBACK_INFORMATION *CallbackParam*
17);

18 The *WriteMiniDump* API writes a user-mode minidump to the file specified
19 by *hFile*. The parameter *hProcess* is a process handle with full read and query
20 permissions. *ProcessId* is a process identifier for the process. The *hFile* parameter
21 is a handle to a file where the user minidump information is written. *DumpType* is
22 used to identify the type of dump to generate. In other words, different types of
23 minidumps supporting different feature sets can be supported. The
24 *ExceptionParam* is an optional pointer to a structure describing the exception in
25 the client that caused the mini-dump to be generated. If this parameter is NULL,
no exception information will be written to the minidump file. The *UserStream* is
an optional pointer to an array of USER_DATA_ENTRY structures. A user data
entry is optional user information that is written to the dump file. The content of
the user information is arbitrary and is never interpreted by the *WriteMiniDump*

routine. *UserStream* should be NULL if there is no optional user information. *CallbackParam* is an optional pointer to a user minidump callback data pointer and routine that *WriteMiniDump* can callback with extended information about the dump. The callback allows the caller of *WriteMiniDump* to completely control the granularity of information written to a user minidump file. If this pointer is NULL, no callbacks are performed.

The return values of the *WriteMiniDump* API are TRUE (1) upon success, and FALSE (0) upon failure.

The *ReadMiniDump* API is:

```
BOOL
WINAPI
ReadMinidumpStream (
    IN PVOID BaseOfDump,
    IN ULONG StreamNumber,
    OUT PMINIDUMP_DIRECTORY * Dir, OPTIONAL
    OUT PVOID * StreamPointer, OPTIONAL
    OUT ULONG * StreamSize OPTIONAL
);
```

The *ReadMiniDump* API reads a field from a user-mode minidump. Since the user minidump file specification is well defined (e.g., see Fig. 5 below) the *ReadMiniDump* API can be considered strictly a convenience.

Here, *BaseOfDump* is the header of the mapped user minidump file, *StreamNumber* is the type of data to read from the mini-dump file, and *Dir*, on return, contains a pointer to the user minidump directory of type *DataType* file format. *StreamPointer*, on return, contains a pointer to the beginning of the minidump stream of type *DataType*. The type of data that this points to depends on the type of data requested in the *DataType* parameter. *StreamSize*, on return, contains the size of the stream.

1 The return values of the *ReadMiniDump* API are simply; TRUE (1) upon
2 success, and FALSE (0) upon failure.

3 With this exemplary implementation in mind, a format for a user minidump
4 file will now be described with reference to Fig. 5. The user minidump file 400
5 includes a header section 402, a directory section 404 and a data section 406.
6 These sections are indexed using RELATIVE VIRTUAL ADDRESSES (RVAs),
7 which describe the location of a data member within a file. An RVA is an offset
8 from the beginning of a file. In Fig. 5, RVA 408 points to the first directory in
9 section 404, and RVAs 410(a-c) point to respective data 412(a-c) in section 406.

10 Since it is important for a user minidump file from one computer
11 architecture to be readable on a computer of a different architecture, all structures
12 in user minidump file 400 are therefore of the same size, regardless of the
13 computer system they are generated on.

14 As shown in Fig. 5, the user minidump file format is similar to the Portable
15 Executable image file format in that it specifies a set of directories that then point
16 to the data (e.g., see *Microsoft Portable Executable Common Object File Format*
17 *Specification Revision 6.0*: Visual C++ Business Unit, Microsoft Corporation,
18 February 1999). Each directory entry in section 404 specifies the data type, the
19 size of the data and an RVA 410 to where in the user minidump file 400 the data is
20 located. In this example, only one directory of a specific type is allowed per file.

21 Those skilled in the art will recognize that the above described methods and
22 arrangements are flexible in that a variety of operating systems and applications
23 can be supported. The amount of resulting kernel minidump information or user
24 minidump information can be increased or decreased as needed. For example, in
25 certain implementations, the user minidump file can be reduced even further to

1 only include the faulting thread, faulting thread's callstack and context, module
2 list, and reason for the crash. This can further reduce the user minidump file to
3 several dozen bytes.

4 Additionally, in certain implementations, information can be added to the
5 user minidump file to make it more usable by debuggers. One limitation of the
6 user minidump process as described above is that it does not store any global
7 variables or dynamically allocated memory. If needed, however, global variables
8 could be saved by adding global data from the faulting process or, more
9 sophisticatedly, by adding only selected regions of global data. These selected
10 regions of global data can be selected by a custom-written application that has
11 knowledge about where certain data is stored in the specific application. Also,
12 selected data pages from the process could also be stored as needed to view active
13 heap variables. The user minidump file format of Fig. 5 has the built-in flexibility
14 to add these data while maintaining compatibility with previous versions.

15 Although some preferred embodiments of the various methods and
16 arrangements of the present invention have been illustrated in the accompanying
17 Drawings and described in the foregoing Detailed Description, it will be
18 understood that the invention is not limited to the exemplary embodiments
19 disclosed, but is capable of numerous rearrangements, modifications and
20 substitutions without departing from the spirit of the invention as set forth and
21 defined by the following claims.